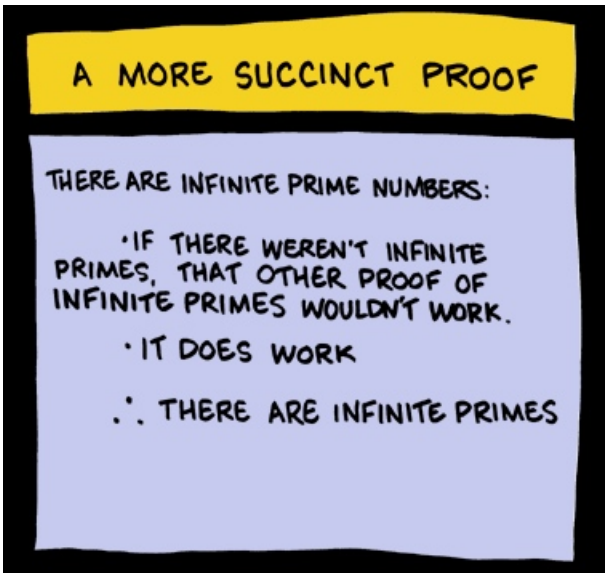
CS140 - Assignment 2

Due: Sunday, Feb. 4 at 10pm



http://www.smbc-comics.com/index.php?db=comics&id=1099

This problem set should be done with a *different* partner than you worked with last week. If you would like help finding a partner, reach out ASAP (not later than Thursday).

You must use LaTeX to format your solution; one person should upload the pdf

1. **[5 points]** Algorithm A has a running time described by the recurrence $T(n) = 7T(n/2)+n^2$. A competing algorithm B has a running time described by the recurrence $T(n) = aT(n/4) + n^2$. What is the largest integer value for a such that B is asymptotically faster than A? Explain your answer. (Hint: use the master method.)

2. **[10 points]** 3-part sort

   Consider the following sorting algorithm: First sort the first two-thirds of the elements in the array. Next sort the last two thirds of the array. Finally, sort the first two thirds again. Notice that this algorithm does not allocate any extra memory; all the sorting is done inside array A. Here's the code:

```
ThreeSort (A,i,j)
    if A[i] > A[j]
      swap A[i] and A[j]
```

if $i + 1 \geq j$
  return
$k = \lfloor (j - i + 1)/3 \rfloor$.
ThreeSort$(A, i, j - k)$ **Comment:** Sort first two-thirds.
ThreeSort$(A, i + k, j)$ **Comment:** Sort last two thirds.
ThreeSort$(A, i, j - k)$ **Comment:** Sort first two-thirds again!

(a) Give an informal but convincing explanation (not a rigorous proof by induction) of why the approach of sorting the first two-thirds of the array, then sorting the last two-thirds of the array, and then sorting again the first two-thirds of the array yields a sorted array. A few well-chosen sentences should suffice here.

(b) Find a recurrence relation for the worst-case running time of ThreeSort.

(c) Solve the recurrence relation

(d) How does the worst-case running time of ThreeSort compare with the worst-case running times of Insertion Sort, Selection Sort, and Mergesort?

3. **[7 points]** Halfsies

   In some situations, there is not a natural ordering to the data so we can't sort it, but we can check equality (e.g. images). Given an array of elements A, we would like to determine if there exists a value that occurs in more than half of the entries of the array. If so, return that value, otherwise, return *null*. Assume you can only check equality of elements in the array which takes time $O(1)$.

   Describe a divide-and-conquer algorithm whose running time is asymptotically better than $\Theta(n^2)$. Provide pseudocode and/or a clear English description of your algorithm and state the worst case running time of your solution (in terms of Θ or O where appropriate) *with justification.*

4. **[10 points]** Given a *sorted* list of unique integers A[1...n], determine if an entry exists such that $A[i] = i$. If an entry exists, return the index, otherwise, return *null*. (Hint: You can do better than $O(n)$. Think divide-and-conquer.) Write **pseudocode** to solve this problem and and *state the worse case running time* (in terms of Θ or O where appropriate). You will be graded on the efficiency of your solutions.

5. **[25 points]** Stock Market Problem

   You're given an array of numbers representing a stock's prices over n days. You goal is to identify the longest consecutive number of days during which the stock's value does not decrease. For example, consider the stock values below:

Day:	1	2	3	4	5	6	7	8
Value:	42	40	45	45	44	43	50	49

   In this example, the length of the longest consecutive non-decreasing run is 3. This run goes from day 2 to day 4.

(a) Briefly describe a very simple "naive" algorithm for this problem and explain why the worst-case running time is $\Theta(n^2)$.

(b) Describe a divide-and-conquer algorithm whose running time is asymptotically better than $\Theta(n^2)$. Provide pseudocode and/or a clear English description of your algorithm. (Note that your algorithm must be a divide-and-conquer algorithm. And yes there are non-divide-and-conquer algorithms that are very, very good!)

Hint: Like writing recursive functions, when trying to come up with a divide-and-conquer solution, assume that your algorithm works correctly on the divided parts. Then, how do you construct your answer to the overall solution?

(c) Analyze the running time of your algorithm: what are tight bounds on the best and worst case behavior?

(Note that next week's assignment will ask you to implement your divide-and-conquer algorithm.)